UNIVERSIDADE FEDERAL DE MINAS GERAIS
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELETRÔNICA

PROJETO RTL
SISTEMA DE COFRE DIGITAL

DAVI PAULO VILELA MOURA
LUIS HENRIQUE CORDEIRO RODRIGUES
VINICIUS MOREIRA FARIA

Belo Horizonte - MG
2025

Sumário

1. Introdução

Este trabalho tem como objetivo a implementação de um sistema de tranca digital para cofres, baseado em uma senha de 16 bits. O sistema foi projetado para oferecer segurança e praticidade, permitindo ao usuário configurar, validar e redefinir a senha conforme necessário.

O funcionamento do sistema é baseado em três modos operacionais principais:

1. **Modo de Configuração:** O usuário define uma nova senha para o cofre, que será armazenada no sistema e utilizada nas próximas tentativas de acesso.
2. **Modo de Tentativa:** Após a configuração da senha, o usuário pode inserir uma combinação para tentar destrancar o cofre. Caso a senha digitada seja correta, o sistema autoriza a abertura. Se a senha inserida for incorreta, um contador de tentativas é ativado, limitando o número de erros consecutivos antes do bloqueio temporário do sistema.
3. **Modo de Reset:** Para situações em que seja necessário restaurar o sistema, o modo de reset redefine a senha para um valor padrão de fábrica, representado pelo código binário 0000000000000000.

A implementação do sistema seguiu um processo estruturado, iniciando com a definição dos requisitos e das funcionalidades desejadas. Em seguida, foi realizada a modelagem do sistema a partir de uma FSM de alto nível, capturando a idéia principal do projeto. Com isso, foi contemplando o projeto do caminho de dados e a criação de uma Máquina de Estados de baixo nível para modelar as mudanças de estado. Por fim, implementamos o código em VHDL. Todas as etapas foram desenvolvidas seguindo a metodologia **Register Transfer Level (RTL)**, garantindo uma abordagem eficiente e organizada para o desenvolvimento do circuito digital.

Com essa abordagem, o projeto busca oferecer um sistema confiável e seguro para o controle de acesso a cofres digitais, aplicando conceitos fundamentais de eletrônica digital e engenharia de hardware.

2. Modelagem do Sistema

Foram determinados todos os periféricos com os quais o processador irá interagir para o funcionamento ideal da trava digital, bem como as entradas e saídas. A figura 1 mostra o diagrama de blocos do sistema.

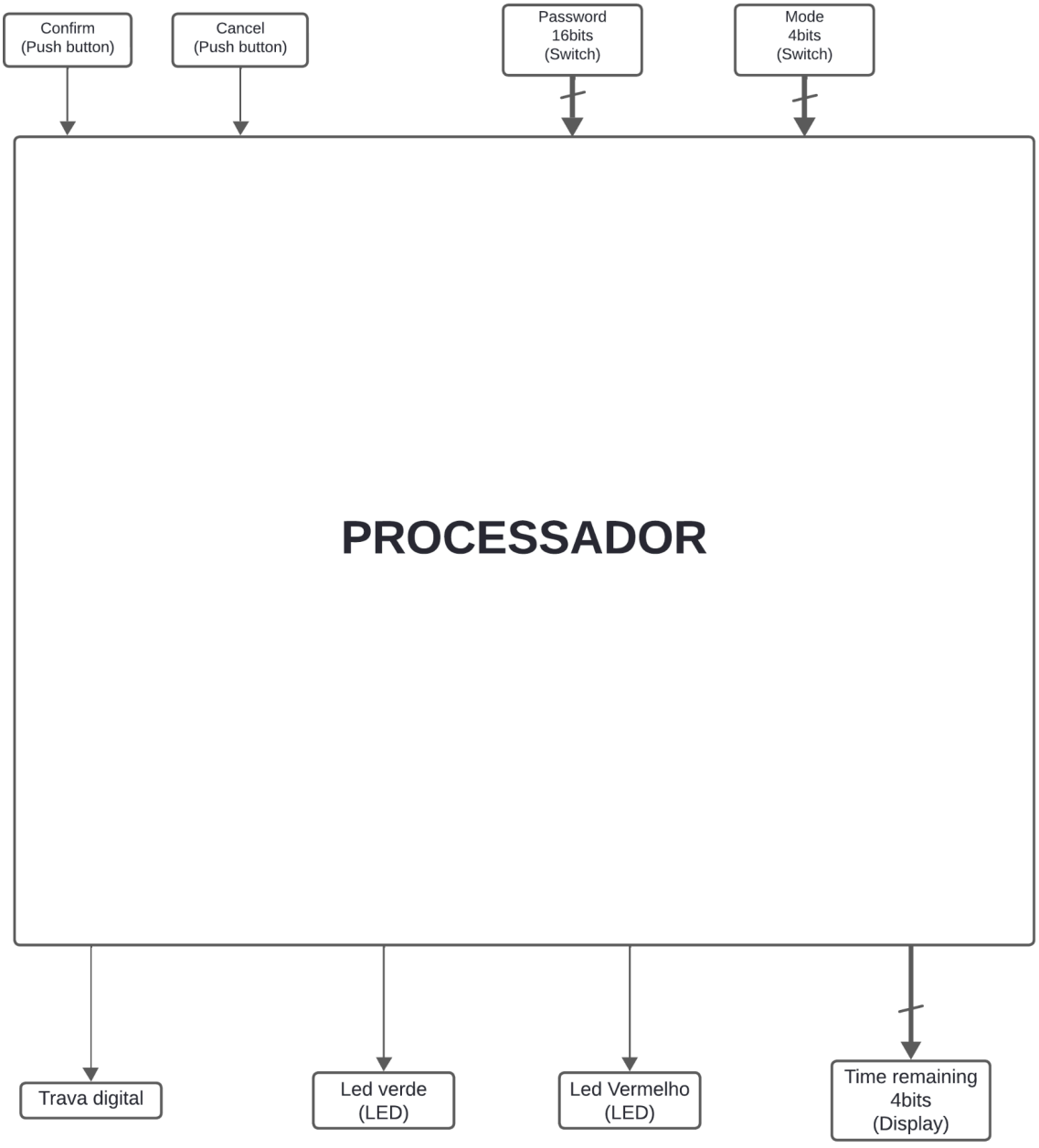

FIGURA 1: Diagrama de blocos do sistema.

A seguir, serão especificadas as funções, entradas, saídas e protocolos de comunicação de cada um dos periféricos, para que se possa projetar o processador a partir de tais especificações.

2.1 Periféricos

Botão Confirm:

Descrição: push button com a finalidade de confirmação.
Após o usuário "digitar" sua senha, o botão deve ser pressionado para que o sistema interprete aquela entrada como uma tentativa. Condicionando-o desta forma, inibe-se que tentativas sejam consideradas com base nas bordas de clock. O botão também tem funcionalidades específicas atreladas a transições de estados, como será apresentado posteriormente na Controladora.

Sinal: Entrada digital de 1 bit. Nível lógico alto quando pressionado.

Botão Cancel:

Descrição: push button com a finalidade de cancelamento de determinadas funcionalidades.
A exemplo do botão de confirmação, também tem funcionalidades específicas atreladas a transições de estados, como será apresentado posteriormente na controladora.

Sinal: Entrada digital de 1 bit. Nível lógico alto quando pressionado.

Password:

Descrição: Se trata de um valor binário de 16 bits. Para fins de simplificação do funcionamento e da compreensão do sistema, são utilizados 16 switches, sendo cada um responsável por um bit. Este valor representa a senha da qual o usuário está dando entrada no sistema. Seja para uma tentativa de abrir o cofre, seja para configurá-la como a nova senha do sistema.

Sinal: Entrada digital de 16 bits. Nível lógico de cada bit alto quando seu respectivo switch está ativado.

Trava:

Descrição: Iremos considerar a Trava digital do cofre como um sinal binário de 1 bit, que em nível lógico baixo denota cofre destrancado e em nível lógico alto denota cofre trancado.

Sinal: Saída digital de 1 bit. Aceso em nível lógico alto.

Led Vermelho:

Descrição: O led vermelho é ativado quando o usuário erra a senha ou quando o sistema entra em bloqueio(3 erros de senha).

Sinal: Saída digital de 1 bit.

Led Verde:

Descrição: O led verde é ativado quando o usuário acerta a senha destrancando o cofre.

Sinal: Saída digital de 1 bit. Aceso em nível lógico alto.

Timing remaining:

Descrição: Saída para um display que denota o tempo restante para que o usuário realize uma nova tentativa de senha. Após errar 3 vezes a senha, de forma consecutiva, o sistema entra em modo de bloqueio por X segundos até que o usuário possa realizar uma nova tentativa.

Sinal: Saída digital de 4 bits.

2.2 Projeto do Processador - Metodologia RTL

2.2.1 FSM de alto nível

A figura 2 mostra a FSM de alto nível que o processador deverá implementar e a seguir o procedimento é explicado. Foi utilizada a convenção de que quando uma saída não é explicitada no estado da FSM, ela está sendo representada como nível lógico baixo, com exceção de quando o nível lógico baixo é necessário para o estado.

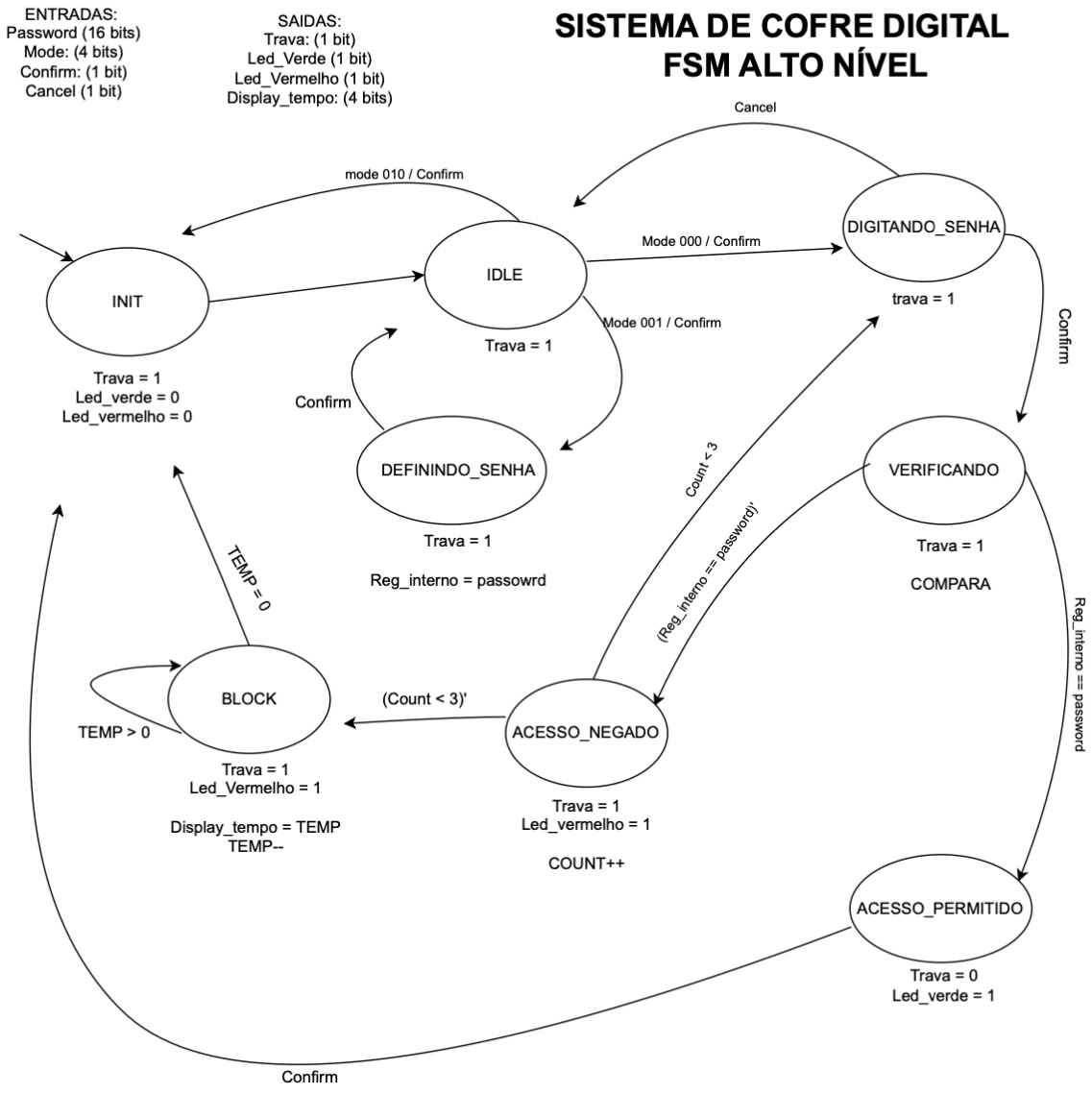

Figura 2: FSM de alto nível

Estados da FSM:

- **INIT (Estado Inicial):**

  Trava = 1: O cofre está travado.

  Neste estado, o sistema reseta seus componentes e segue para o próximo estado. aguarda uma entrada do usuário para iniciar uma operação (definir senha ou digitar senha).

- **IDLE (Estado Inicial):**

  Trava = 1: O cofre está travado.

  Neste estado, o sistema aguarda a confirmação da escolha do modo de operação para prosseguir para o próximo estado.

- **DEFININDO_SENHA (Definir Senha):**

  Trava = 1: O cofre permanece travado.

  Reg_interno = password: A senha digitada é armazenada em um registrador interno.

  Este estado é alcançado quando o usuário está definindo uma nova senha. A senha é armazenada para futuras verificações.

- **DIGITANDO_SENHA (Digitando Senha):**

  Trava = 1: O cofre permanece travado.

  Neste estado, o usuário está digitando a senha para tentar abrir o cofre. O sistema aguarda a confirmação da senha.

- **VERIFICANDO (Verificando Senha):**

  Trava = 1: O cofre permanece travado.

  COMPARA: O sistema compara a senha digitada com a senha armazenada no registrador interno.

Dependendo do resultado da comparação, o sistema transita para **ACESSO_PERMITIDO** ou **ACESSO_NEGADO**.

- **ACESSO_PERMITIDO (Acesso ao Cofre):**

  Trava = 0: O cofre é destravado.

  Led_Verde = 1: O LED verde é aceso, indicando que o acesso foi permitido.

  Ao atingir esse estado estado, o sistema destrava o cofre e espera pela confirmação do usuário para o reset do sistema.

- **ACESSO_NEGADO (Acesso Negado):**

  Trava = 1: O cofre permanece travado.

  Led_Vermelho = 1: O LED vermelho é aceso, indicando que o acesso foi negado.

  COUNT++: Um contador é incrementado para rastrear o número de tentativas falhas.

  Se o número de tentativas falhas atingir um limite (por exemplo, 3), o sistema irá transitar para um estado de bloqueio temporário.

- **BLOCK (Bloqueado):**

  Trava = 1: O cofre permanece travado.

  Led_Vermelho = 1: O LED vermelho é aceso.

  Ao chegar nesse estado, um temporizador é acionado e um sinal de bloqueio é enviado à controladora, o sinal permanece em nível lógico alto enquanto o contador for diferente de 0. Ao zerar o temporizador, o sinal de bloqueio volta a ter nível baixo, liberando o sistema para retornar ao estado IDLE.

Transições entre Estados:

- **IDLE → DEFINIDO_SENHA:** Ocorre quando o usuário seleciona o modo de definir senha e confirma.
- **IDLE → DIGITANDO_SENHA:** Ocorre quando o usuário seleciona o modo de digitar senha e confirma.

- **DIGITANDO_SENHA → VERIFICANDO:** Ocorre quando o usuário confirma a senha digitada.
- **VERIFICANDO → ACESSO_PERMITIDO:** Ocorre se a senha digitada for correta.
- **VERIFICANDO → ACESSO_NEGADO:** Ocorre se a senha digitada for incorreta.
- **ACESSO_NEGADO → BLOCK:** Ocorre após 3 tentativas falhas.
- **BLOCK → IDLE:** Ocorre quando o tempo de bloqueio expira.

Comportamento do Sistema:

- O sistema começa no estado **IDLE**, onde aguarda a interação do usuário.
- Se o usuário escolher definir uma nova senha, o sistema transita para **DEFINIDO_SENHA**, onde a senha é armazenada.
- Se o usuário escolher digitar uma senha, o sistema transita para **DIGITANDO_SENHA** e, após a confirmação, para **VERIFICANDO**, onde a senha é comparada.
- Se a senha estiver correta, o sistema destrava o cofre e acende o LED verde (**ACESSO_PERMITIDO**).
- Se a senha estiver incorreta, o sistema acende o LED vermelho e incrementa o contador de tentativas falhas (**ACESSO_NEGADO**).
- Após várias tentativas falhas, o sistema entra em estado de bloqueio (**BLOCK**), onde o cofre permanece travado por um tempo determinado.

2.2.2 Caminho de dados

A figura 3 mostra o datapath implementado. O diagrama interno de cada bloco operacional será explicitado adiante.

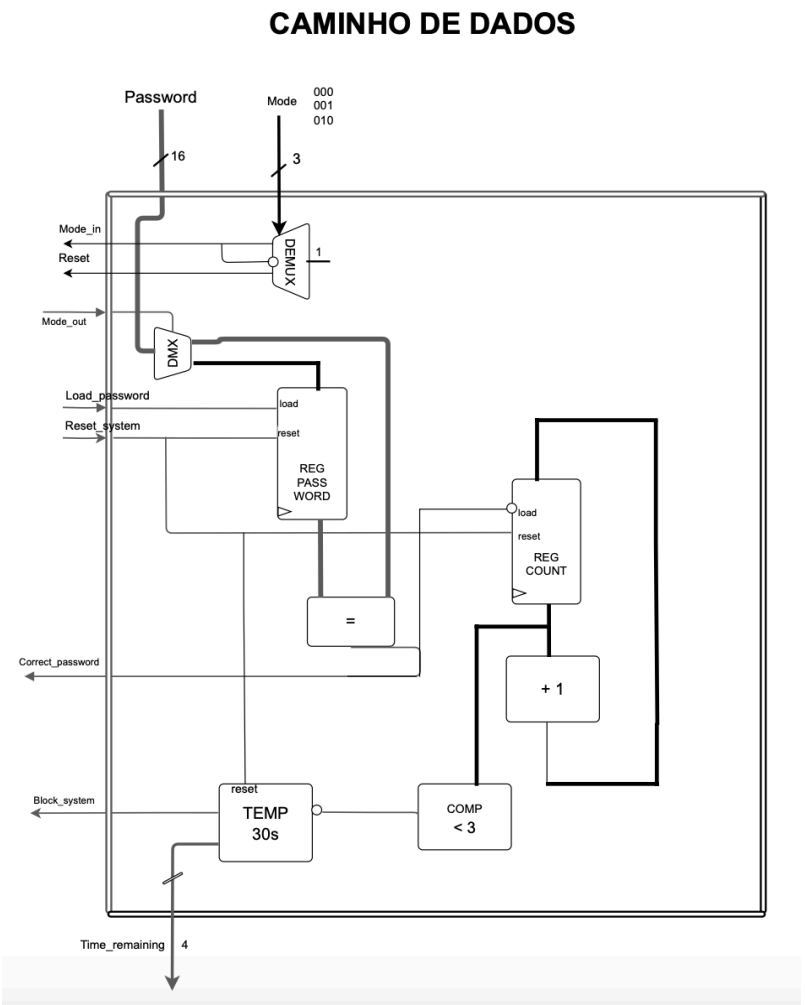

Figura 3: Caminho de dados

Blocos Operacionais utilizados:

1 - Demultiplexador: Permite selecionar um caminho para a entrada de dados

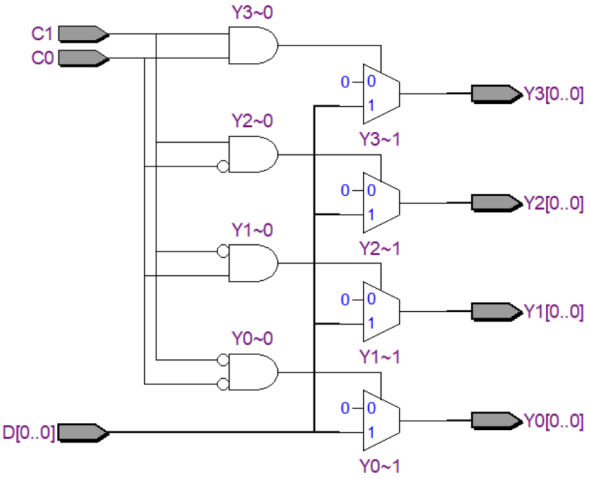

Figura 4: RTL viewer do bloco demultiplexador de 2 bits de escolha.

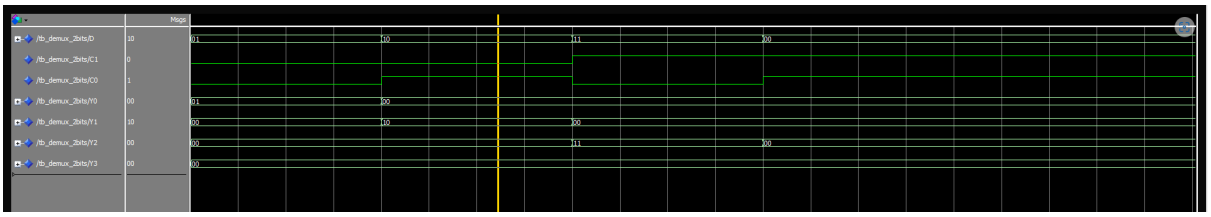

Figura 5:Simulação do bloco demultiplexador de 2 bits de escolha.

2 - Registrador: Armazena um número binário de *n* bits internamente no sistema

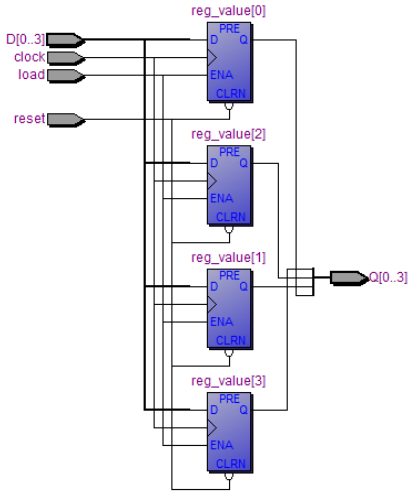

Figura 6: Exemplo de Registrador de 4 bits.

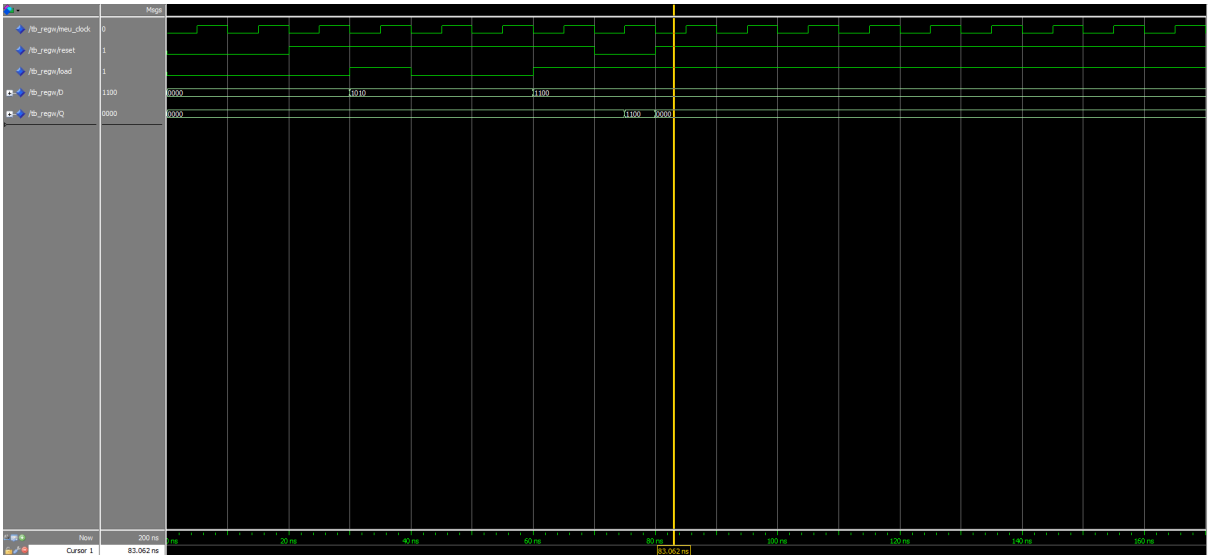

Figura 7: Simulação do bloco Registrador de 4 bits.

3 - Comparador: Compara duas entradas de 16 bits. Saída “igual” = 1 caso sejam iguais, caso contrário, saída “igual” = 0.

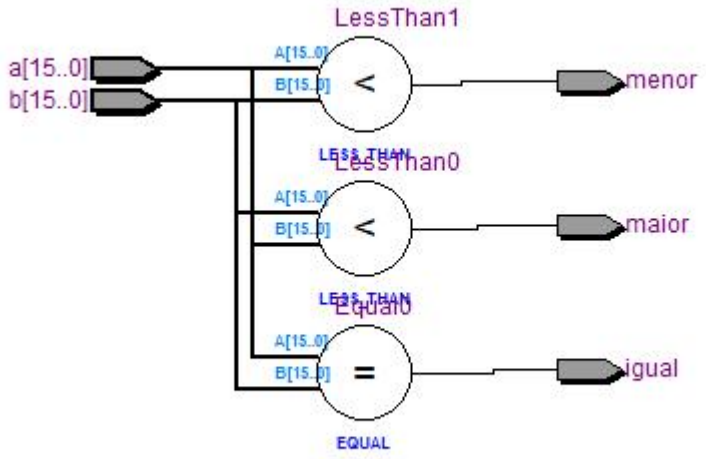

Figura 8: RTL viewer do bloco comparador de 16 bits.

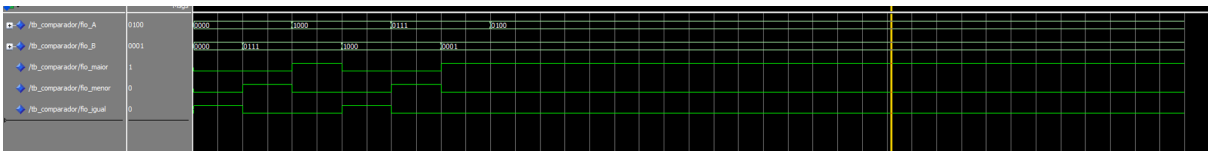

Figura 9: Simulação do bloco comparador de 16 bits.

4 - Full adder: Soma duas entradas de 4 bits, sendo uma delas sempre 0001. O bloco funciona como um incrementador a cada load do registrador com a finalidade de contar as tentativas de senhas erradas.

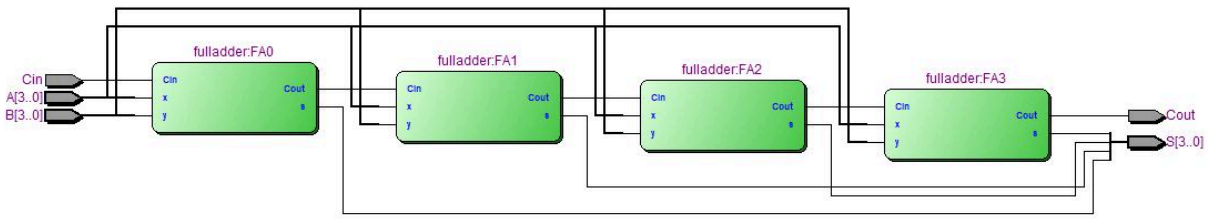

Figura 10: RTL viewer do bloco somador de 4 bits.

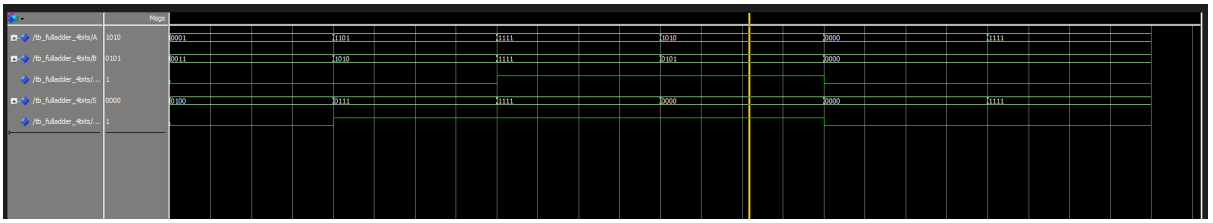

Figura 11: Simulação do bloco somador de 4 bits.

5 - **Timer:**

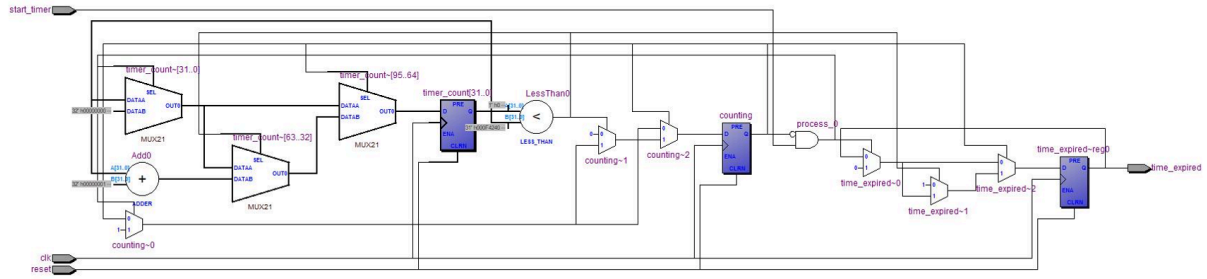

Figura 12: RTL viewer do bloco temporizador.

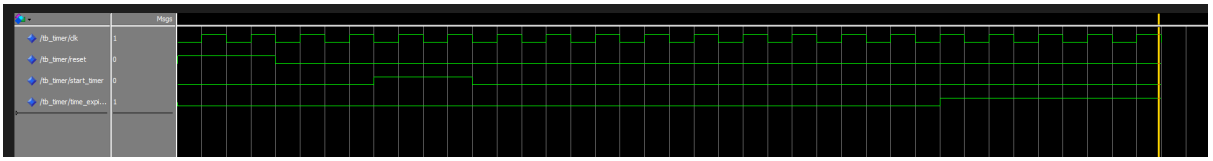

Figura 13: Simulação do bloco temporizador.

2.3 Projeto do Bloco de Controle

2.3.1 Máquina de Estados de baixo Nível (Controladora)

A figura mostra o diagrama de estados da controladora, implementada a partir da FSM de Alto Nível (Figura 2).

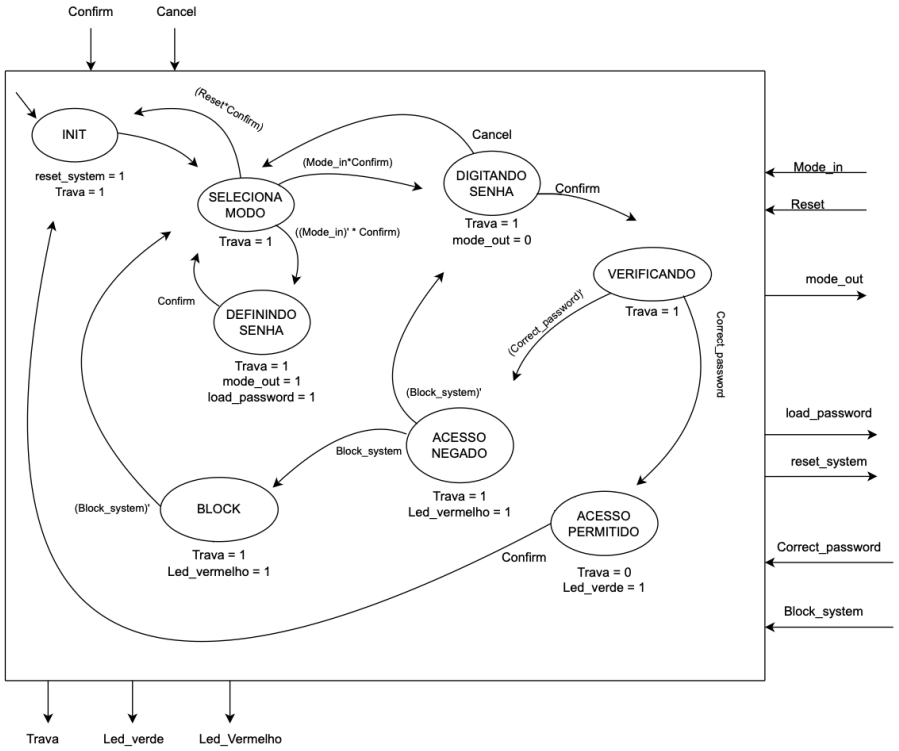

Figura 14 : FSM da Controladora

2.3.1 Conexão da controladora ao Datapath

A figura mostra o diagrama "duas caixas” conectando o Bloco de Controle ao Caminho de Dados.

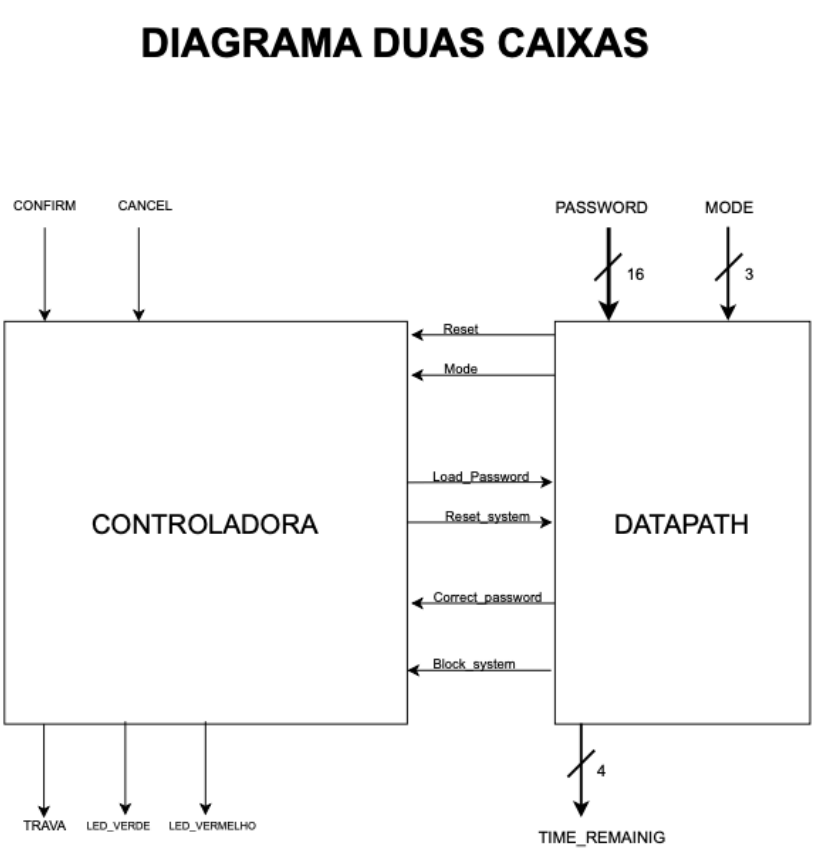

Figura 15: Diagrama 2 caixas do processador.

3 Código VHDL e Simulação do Sistema

Nessa seção será apresentado o código de todos os Componentes utilizados nesse projeto, bem como os arquivos do Caminho de Dados, da Controladora e da conexão entre os dois blocos.

Registrador de 16 bits:

Foi usado um registrador de 16 bits para armazenar a senha internamente no sistema.

Codigo em VHDL:

```
LIBRARY IEEE;
USE ieee.std_logic_1164.ALL;

ENTITY tb_RegW IS
END tb_RegW;

ARCHITECTURE behavior OF tb_RegW IS

    -- Component Declaration
    COMPONENT RegW
        GENERIC (
            W : INTEGER := 16
        );
        PORT (
            clock : IN STD_LOGIC;
            reset : IN STD_LOGIC;
            load : IN STD_LOGIC;
            D : IN STD_LOGIC_VECTOR(W - 1 DOWNTO 0);
            Q : OUT STD_LOGIC_VECTOR(W - 1 DOWNTO 0)
        );
    END COMPONENT;

    -- Signals for the testbench
    SIGNAL meu_clock : STD_LOGIC := '0';
    SIGNAL reset : STD_LOGIC;
    SIGNAL load : STD_LOGIC;
    SIGNAL D : STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL Q : STD_LOGIC_VECTOR(3 DOWNTO 0);

```

```
30  BEGIN
31
32      -- Instantiate the RegW with W=4
33      uut : RegW
34      GENERIC MAP(
35          W => 16
36      )
37      PORT MAP(
38          clock => meu_clock,
39          reset => reset,
40          load => load,
41          D => D,
42          Q => Q
43      );
44
45      -- Clock stimulus generation
46      meu_clock <= NOT meu_clock AFTER 5 ns;
47
```

Demultiplexador:

O demux foi usado em duas ocasiões no projeto, uma para selecionar o modo de operação do sistema e outra para selecionar o caminho da senha digitada pelo usuário, podendo ir para o **Registrador** interno ou diretamente para o **Comparador**

Codigo em VHDL:

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY demux_2bits IS
    GENERIC (
        DATA_WIDTH : INTEGER := 1 -- Tamanho padrão de D
    );
    PORT (
        D : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0); -- Entrada de dados (tamanho genérico)
        C1 : IN STD_LOGIC; -- Entrada de controle 1
        C0 : IN STD_LOGIC; -- Entrada de controle 0
        Y0 : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0); -- Saída 0 (tamanho genérico)
        Y1 : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0); -- Saída 1 (tamanho genérico)
        Y2 : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0); -- Saída 2 (tamanho genérico)
        Y3 : OUT STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0) -- Saída 3 (tamanho genérico)
    );
END demux_2bits;

ARCHITECTURE Behavioral OF demux_2bits IS
BEGIN
    -- Lógica do demux
    Y0 <= D WHEN (C1 = '0' AND C0 = '0') ELSE
        (OTHERS => '0'); -- Saída Y0 ativa quando C1=0 e C0=0
    Y1 <= D WHEN (C1 = '0' AND C0 = '1') ELSE
        (OTHERS => '0'); -- Saída Y1 ativa quando C1=0 e C0=1
    Y2 <= D WHEN (C1 = '1' AND C0 = '0') ELSE
        (OTHERS => '0'); -- Saída Y2 ativa quando C1=1 e C0=0
    Y3 <= D WHEN (C1 = '1' AND C0 = '1') ELSE
        (OTHERS => '0'); -- Saída Y3 ativa quando C1=1 e C0=1
END Behavioral;
```

Comparador:

O Comparador foi usado num primeiro momento para fazer a comparação entre a senha armazenada no sistema e a senha digitada pelo usuário, e posteriormente para fazer a comparação do numero de tentativas erradas.

Codigo VHDL:

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY comparador IS
    GENERIC (
        DATA_WIDTH : NATURAL := 16 -- Largura dos vetores A e B
    );
    PORT (
        a : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
        b : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
        maior : OUT STD_LOGIC;
        menor : OUT STD_LOGIC;
        igual : OUT STD_LOGIC
    );
END comparador;

ARCHITECTURE concorrente OF comparador IS
BEGIN
    -- Comparação considerando valores absolutos
    maior <= '1' WHEN (UNSIGNED(a) > UNSIGNED(b)) ELSE
        '0';
    menor <= '1' WHEN (UNSIGNED(a) < UNSIGNED(b)) ELSE
        '0';
    igual <= '1' WHEN (UNSIGNED(a) = UNSIGNED(b)) ELSE
        '0';
END concorrente;
```

Somador:

O somador foi usado para incrementar o numero de tentativas erradas.

Codigo VHDL:

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   entity fulladder is
5       port (
6            Cin : in std_logic;
7            x   : in std_logic;
8            y   : in std_logic;
9            s   : out std_logic;
10           Cout : out std_logic
11          );
12  end fulladder;
13  architecture RTL OF fulladder is
14  begin
15      s <= x XOR y XOR Cin;
16      Cout <= (x AND y) OR (Cin AND x) OR (Cin AND y);
17  end RTL ;
```

Timer:

O timer foi usado para contar o tempo de bloqueio do sistema após as 3 tentativas erradas

Código VHDL:

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

-- 5. Timers
ENTITY timer IS
    GENERIC (
        W : NATURAL := 1000000 -- Valor padrão (ciclos de clk) para o timer
    );
    PORT (
        clk : IN STD_LOGIC;
        reset : IN STD_LOGIC;
        start_timer : IN STD_LOGIC;
        time_expired : OUT STD_LOGIC
    );
END timer;

ARCHITECTURE Behavioral OF timer IS
    CONSTANT TIME_LIMIT : INTEGER := W; -- Define o limite de tempo
    SIGNAL timer_count : INTEGER := 0;  -- Contador interno
    SIGNAL counting : STD_LOGIC := '0'; -- Sinal para indicar que o timer está contando
BEGIN
    PROCESS (clk, reset)
    BEGIN
        IF reset = '1' THEN
            -- Reseta o contador e desativa o sinal de tempo expirado
            timer_count <= 0;
            time_expired <= '0';
            counting <= '0';
        ELSIF rising_edge(clk) THEN
            -- Detecta a borda de subida de start_timer para iniciar a contagem
            IF start_timer = '1' AND counting = '0' THEN
                counting <= '1'; -- Inicia a contagem
                timer_count <= 0; -- Reinicia o contador
                time_expired <= '0'; -- Desativa o sinal de tempo expirado
            END IF;
            -- Se o timer estiver contando, incrementa o contador
            IF counting = '1' THEN
                IF timer_count < TIME_LIMIT THEN
                    timer_count <= timer_count + 1; -- Incrementa o contador
                ELSE
                    time_expired <= '1'; -- Ativa o sinal de tempo expirado
                    counting <= '0'; -- Para a contagem
                END IF;
            END IF;
        END IF;
    END PROCESS;
END Behavioral;
```

Datapath:

O caminho de dados implementa a mudança que os dados sofrem no fluxo do sistema.

Código VHDL:

```
40      COMPONENT RegW
41          GENERIC (
42              W : INTEGER := 16 -- Largura padrão do registrador
43          );
44          PORT (
45              clock : IN STD_LOGIC;
46              reset : IN STD_LOGIC; -- Reset assíncrono ativo em '1'
47              load : IN STD_LOGIC; -- Load síncrono ativo em '1'
48              D : IN STD_LOGIC_VECTOR(W - 1 DOWNTO 0);
49              Q : OUT STD_LOGIC_VECTOR(W - 1 DOWNTO 0)
50          );
51      END COMPONENT;
52
53      COMPONENT comparador
54          GENERIC (
55              DATA_WIDTH : NATURAL := 16 -- Largura dos vetores A e B
56          );
57          PORT (
58              a : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
59              b : IN STD_LOGIC_VECTOR(DATA_WIDTH - 1 DOWNTO 0);
60              maior : OUT STD_LOGIC;
61              menor : OUT STD_LOGIC;
62              igual : OUT STD_LOGIC
63          );
64      END COMPONENT;
65
66      COMPONENT timer
67          GENERIC (
68              W : NATURAL := 1000000 -- Valor padrão (ciclos de clk) para o timer
69          );
70          PORT (
71              clk : IN STD_LOGIC;
72              reset : IN STD_LOGIC;
73              start_timer : IN STD_LOGIC;
74              time_expired : OUT STD_LOGIC
75          );
76      END COMPONENT;
```

```
78      COMPONENT fulladder_4bits
79          PORT (
80              A : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
81              B : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
82              Cin : IN STD_LOGIC;
83              S : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
84              Cout : OUT STD_LOGIC
85          );
86      END COMPONENT;
87
88      -- Sinais internos
89      SIGNAL demux1_out1 : STD_LOGIC_VECTOR(0 DOWNTO 0); -- Saída do demux de 1 bit
90      SIGNAL demux1_out2 : STD_LOGIC_VECTOR(0 DOWNTO 0); -- Saída do demux de 1 bit
91      SIGNAL demux1_out3 : STD_LOGIC_VECTOR(0 DOWNTO 0); -- Saída do demux de 1 bit
92      SIGNAL demux16_out1 : STD_LOGIC_VECTOR(15 DOWNTO 0); -- Saída do demux de 16 bits
93      SIGNAL demux16_out2 : STD_LOGIC_VECTOR(15 DOWNTO 0); -- Saída do demux de 16 bits
94      SIGNAL reg_password_out : STD_LOGIC_VECTOR(15 DOWNTO 0); -- Saída do registrador de senha
95      SIGNAL reg_error_count_out : STD_LOGIC_VECTOR(1 DOWNTO 0); -- Saída do registrador de contagem de erros
96      SIGNAL comp_password_out : STD_LOGIC; -- Saída do comparador de senhas
97      SIGNAL comp_error_count_out : STD_LOGIC; -- Saída do comparador de contagem de erros
98      SIGNAL timer_led_out : STD_LOGIC; -- Saída do timer para piscar o LED
99      SIGNAL timer_block_out : STD_LOGIC; -- Saída do timer para bloquear o sistema
100     SIGNAL counter_out : STD_LOGIC_VECTOR(3 DOWNTO 0); -- Saída do contador de 4 bits
101     SIGNAL counter_reset : STD_LOGIC; -- Sinal de reset do contador
102
```

```
103   BEGIN
104
105       -- Instanciação do demux de 1 bit
106       demux1 : demux_2bits
107       GENERIC MAP(
108           DATA_WIDTH => 1
109       )
110       PORT MAP(
111           D => '1',
112           C1 => mode(1),
113           C0 => mode(0),
114           Y0 => mode_in,
115           Y1 => NOT mode_in,
116           Y2 => reset,
117           Y3 => OPEN
118       );
119
120       -- Instanciação do demux de 16 bits
121       demux16 : demux_2bits
122       GENERIC MAP(
123           DATA_WIDTH => 16
124       )
125       PORT MAP(
126           D => password,
127           C1 => mode_out(1),
128           C0 => mode_out(0),
129           Y0 => demux16_out1,
130           Y1 => demux16_out2,
131           Y2 => OPEN,
132           Y3 => OPEN
133       );
134
135       -- Instanciação do registrador de senha
136       reg_password : RegW
137       GENERIC MAP(
138           W => 16
139       )
140       PORT MAP(
```

```
    PORT MAP(
        clock => clock,
        reset => reset_system,
        load => load_password,
        D => demux16_out2,
        Q => reg_password_out
    );

    -- Instanciação do registrador de contagem de erros
    reg_error_count : RegW
    GENERIC MAP(
        W => 2
    )
    PORT MAP(
        clock => clock,
        reset => reset_system,
        load => NOT comp_password_out,
        D => counter_out(1 DOWNTO 0), -- Usa os 2 bits menos significativos do contador
        Q => reg_error_count_out
    );

    -- Instanciação do comparador de senhas
    comp_password : comparador
    GENERIC MAP(
        DATA_WIDTH => 16
    )
    PORT MAP(
        a => password,
        b => reg_password_out,
        igual => comp_password_out,
        maior => OPEN,
        menor => OPEN
    );

    -- Instanciação do comparador de contagem de erros
    comp_error_count : comparador
    GENERIC MAP(
```

```
    GENERIC MAP(
        DATA_WIDTH => 2
    )
    PORT MAP(
        a => reg_error_count_out,
        b => "11", -- Compara com 3 (em binário: "11")
        igual => comp_error_count_out,
        maior => OPEN,
        menor => OPEN
    );

    -- Instanciação do contador de 4 bits
    counter : fulladder_4bits
    PORT MAP(
        A => "00" & reg_error_count_out, -- Entrada do contador
        B => "0001", -- Incremento de 1
        Cin => '0', -- Sem carry inicial
        S => counter_out,
        Cout => OPEN
    );

    -- Instanciação do timer para piscar o LED
    timer_led : timer
    GENERIC MAP(
        W => 1000000 -- Define o tempo para piscar o LED
    )
    PORT MAP(
        clk => clock,
        reset => reset_system,
        start_timer => comp_error_count_out, -- Inicia o timer quando a contagem de erros atinge 3
        time_expired => timer_led_out
    );
```

```
    -- Instanciação do timer para bloquear o sistema
    timer_block : timer
    GENERIC MAP(
        W => 5000000 -- Define o tempo para bloquear o sistema
    )
    PORT MAP(
        clk => clock,
        reset => reset_system,
        start_timer => comp_error_count_out, -- Inicia o timer quando a contagem de erros atinge 3
        time_expired => timer_block_out
    );

    -- Lógica de saída
    mode_in <= demux1_out(0);
    reset <= reset_system;
    corect_pass <= comp_password_out;
    block_system <= timer_block_out;
    time_remaining <= timer_led_out;

END estrutural;
```

Controladora:

A controladora implementa a lógica de mudança de estados do sistema.

Código VHDl:

new.vhdl

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY controladora IS
    PORT (
        reset : IN STD_LOGIC;
        clock : IN STD_LOGIC;
        mode : IN STD_LOGIC;
        correct_password : IN STD_LOGIC;
        block_system : IN STD_LOGIC;
        confirm : OUT STD_LOGIC;
        cancel : OUT STD_LOGIC;
        load_password : OUT STD_LOGIC;
        reset_system : OUT STD_LOGIC;
        trava : OUT STD_LOGIC
        led_verde : OUT STD_LOGIC
        led_vermelho : OUT STD_LOGIC
    );
END controladora;

ARCHITECTURE arch OF controladora IS
    -- Enumerated type for state machine states
    TYPE state_type IS (INIT, MODE, DEFINE, TYPING, CHECK, DENIED, ALLOWED, BLOCKED);

    -- Signal to hold the current state
    SIGNAL current_state, next_state : state_type;

BEGIN
    -- Process for state transitions (sequential logic)
    PROCESS (clock, reset)
    BEGIN
        IF reset = '1' THEN
            current_state <= INIT; -- Initial state
        ELSIF RISING_EDGE(clock) THEN
            current_state <= next_state;
        END IF;
    END PROCESS;
```

```
39      -- Process for next state logic and output generation (combinational logic)
40      PROCESS (current_state)
41      BEGIN
42          -- Default assignments for outputs
43          reset <= '0';
44          mode <= "00";
45          corect_password <= '0';
46          block_system <= '0';
47          confirm <= '0';
48          cancel <= '0';
49          load_password <= '0';
50          reset_system <= '0';
51          trava <= '1';
52          led_verde <= '0';
53          led_vermelho <= '0';
54          next_state <= current_state; -- Default state
55
56          CASE current_state IS
57              WHEN INIT =>
58                  reset <= '0';
59                  mode <= "00";
60                  corect_password <= '0';
61                  block_system <= '0';
62                  confirm <= '0';
63                  cancel <= '0';
64                  load_password <= '0';
65                  reset_system <= '0';
66                  trava <= '1';
67                  led_verde <= '0';
68                  led_vermelho <= '0';
69                  next_state <= MODE;
70
```

```
71              WHEN MODE =>
72                  reset <= '0';
73                  mode <= "00";
74                  corect_password <= '0';
75                  block_system <= '0';
76                  confirm <= '0';
77                  cancel <= '0';
78                  load_password <= '0';
79                  reset_system <= '0';
80                  trava <= '1';
81                  led_verde <= '0';
82                  led_vermelho <= '0';
83                  next_state <= MODE;
84
85              WHEN DEFINE =>
86                  reset <= '0';
87                  mode <= "00";
88                  corect_password <= '0';
89                  block_system <= '0';
90                  confirm <= '0';
91                  cancel <= '0';
92                  load_password <= '0';
93                  reset_system <= '0';
94                  trava <= '1';
95                  led_verde <= '0';
96                  led_vermelho <= '0';
97                  next_state <= MODE;
```

```
99              WHEN TYPING =>
100                 reset <= '0';
101                 mode <= "00";
102                 corect_password <= '0';
103                 block_system <= '0';
104                 confirm <= '0';
105                 cancel <= '0';
106                 load_password <= '0';
107                 reset_system <= '0';
108                 trava <= '1';
109                 led_verde <= '0';
110                 led_vermelho <= '0';
111                 next_state <= MODE;
112
113             WHEN CHECK =>
114                 reset <= '0';
115                 mode <= "00";
116                 corect_password <= '0';
117                 block_system <= '0';
118                 confirm <= '0';
119                 cancel <= '0';
120                 load_password <= '0';
121                 reset_system <= '0';
122                 trava <= '1';
123                 led_verde <= '0';
124                 led_vermelho <= '0';
125                 next_state <= MODE;
```

```
127             WHEN CHECK =>
128                 reset <= '0';
129                 mode <= "00";
130                 corect_password <= '0';
131                 block_system <= '0';
132                 confirm <= '0';
133                 cancel <= '0';
134                 load_password <= '0';
135                 reset_system <= '0';
136                 trava <= '1';
137                 led_verde <= '0';
138                 led_vermelho <= '0';
139                 next_state <= MODE;
140 
141             WHEN DENIED =>
142                 reset <= '0';
143                 mode <= "00";
144                 corect_password <= '0';
145                 block_system <= '0';
146                 confirm <= '0';
147                 cancel <= '0';
148                 load_password <= '0';
149                 reset_system <= '0';
150                 trava <= '1';
151                 led_verde <= '0';
152                 led_vermelho <= '0';
153                 next_state <= MODE;
```

```
155             WHEN ALLOWED =>
156                 reset <= '0';
157                 mode <= "00";
158                 corect_password <= '0';
159                 block_system <= '0';
160                 confirm <= '0';
161                 cancel <= '0';
162                 load_password <= '0';
163                 reset_system <= '0';
164                 trava <= '1';
165                 led_verde <= '0';
166                 led_vermelho <= '0';
167                 next_state <= MODE;
168 
169             WHEN BLOCKED =>
170                 reset <= '0';
171                 mode <= "00";
172                 corect_password <= '0';
173                 block_system <= '0';
174                 confirm <= '0';
175                 cancel <= '0';
176                 load_password <= '0';
177                 reset_system <= '0';
178                 trava <= '1';
179                 led_verde <= '0';
180                 led_vermelho <= '0';
181                 next_state <= MODE;
182 
183             WHEN OTHERS =>
184                 next_state <= INIT; -- Default state
185         END CASE;
186     END PROCESS;
187 
188 END arch;
```

4. Conclusão

A implementação do sistema de Cofre Digital foi projetada seguindo uma abordagem estruturada de projeto RTL (Register Transfer Level), desde a modelagem conceitual até a definição detalhada dos componentes de hardware e suas interações. A máquina de estados de alto nível definiu o comportamento do sistema, enquanto o diagrama do caminho de dados e o diagrama da controladora detalharam a interligação. A implementação em VHDL exigiu uma abordagem modular, permitindo a divisão do projeto em componentes menores, como entrada de senha, validação, controle de tentativas, temporização de bloqueio e acionamento do motor de travamento. A introdução do temporizador adicionou um nível extra de complexidade, mas foi gerenciada por meio de um contador e comparadores que determinam o tempo de espera antes da liberação do sistema.

Em resumo, o projeto visou fornecer um sistema de segurança funcional e programável, adequado para um cofre digital. A abordagem RTL e a implementação da FSM permitiram um design eficiente e escalável, que pode ser facilmente expandido para incluir novas funcionalidades, como autenticação multifator ou integração com sistemas externos. A próxima etapa envolve a síntese e testes no Quartus II, garantindo que o hardware se comporte conforme especificado e validando a robustez do design implementado.

5. Referências

- **Frank Vahid: Sistemas Digitais, projeto, otimização e HDLs, 2008.**